JOURNAL OF SCHEDULING
J. Sched. **1**, 31–54 (1998)

SCHEDULING A BATCHING MACHINE

PETER BRUCKER[1], ANDREI GLADKY[2], HAN HOOGEVEEN[3], MIKHAIL Y. KOVALYOV[2], CHRIS N. POTTS[4,*], THOMAS TAUTENHAHN[5] AND STEEF L. VAN DE VELDE[6]

[1] *FB Mathematik/Informatik, Universität Osnabrück, D-49069 Osnabrück, Germany*
[2] *Institute of Engineering Cybernetics, 220012 Minsk, Belarus*
[3] *Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 5600, Eindhoven, The Netherlands*
[4] *Faculty of Mathematical Studies, University of Southampton, Southampton SO17 1BJ, U.K.*
[5] *Fakultät für Mathematik, Otto-von-Guericke-Universität Magdeburg, Postfach 4120, D-39016 Magdeburg, Germany*
[6] *Rotterdam School of Management, Erasmus University, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands*

ABSTRACT

We address the problem of scheduling n jobs on a batching machine to minimize regular scheduling criteria that are non-decreasing in the job completion times. A batching machine is a machine that can handle up to b jobs simultaneously. The jobs that are processed together form a batch, and all jobs in a batch start and complete at the same time. The processing time of a batch is equal to the largest processing time of any job in the batch. We analyse two variants: the unbounded model, where $b \geqslant n$; and the bounded model, where $b < n$.

For the unbounded model, we give a characterization of a class of optimal schedules, which leads to a generic dynamic programming algorithm that solves the problem of minimizing an arbitrary regular cost function in pseudopolynomial time. The characterization leads to more efficient dynamic programming algorithms for specific cost functions: a polynomial algorithm for minimizing the maximum cost, an $O(n^3)$ time algorithm for minimizing the number of tardy jobs, an $O(n^2)$ time algorithm for minimizing the maximum lateness, and an $O(n \log n)$ time algorithm for minimizing the total weighted completion time. Furthermore, we prove that minimizing the weighted number of tardy jobs and the total weighted tardiness are NP-hard problems.

For the bounded model, we derive an $O(n^{b(b-1)})$ time dynamic programming algorithm for minimizing total completion time when $b > 1$; for the case with m different processing times, we give a dynamic programming algorithm that requires $O(b^2 m^2 2^m)$ time. Moreover, we prove that due date based scheduling criteria give rise to NP-hard problems. Finally, we show that an arbitrary regular cost function can be minimized in polynomial time for a fixed number of batches. © 1998 John Wiley & Sons, Ltd.

KEY WORDS: scheduling; batching machine; complexity; dynamic programming

1. INTRODUCTION

The problem of scheduling n independent jobs on a batching machine to minimize regular scheduling criteria is considered. A *batching machine* or *batch processing machine* is a machine that can

* Correspondence to: Chris N. Potts, Faculty of Mathematical Studies, University of Southampton, Southampton SO17 1BJ, U.K. E-mail: cnp@maths.soton.ac.uk

Contact/grant sponsor: INTAS; Contract/grant numbers: 93-257, 93-257-Ext
Contact/grant sponsor: Deutscher Akademischer Austauschdienst

CCC 1094–6136/98/010031–24$17.50

process up to b jobs simultaneously, and a *regular* scheduling criterion is one that is non-decreasing in the job completion times. The jobs that are processed together form a *batch*. Specifically, we are interested in the so-called *burn-in model*, in which the processing time of a batch is equal to the maximum processing time of any job assigned to it. All jobs contained in the same batch start and complete at the same time, since the completion time of a job is equal to the completion time of the batch to which it belongs. This model is motivated by the problem of scheduling burn-in operations for large-scale integrated circuit manufacturing; see Reference 1.

Webster and Baker [2] present an overview of algorithms and complexity results for scheduling batch processing machines. They distinguish three types of models: the burn-in model; the model in which the processing time of a batch is equal to the sum of the processing times of its jobs (see also Reference 3); and the model in which the processing time of a batch is a constant, independent of the jobs it contains (see also Reference 4).

We analyse two variants of the burn-in model: the *unbounded model*, in which $b \geqslant n$ so that there is effectively no upper bound on the number of jobs that can be processed in the same batch; and the *bounded model*, in which b is a constant smaller than n so that there is a restrictive upper bound. The unbounded model arises, for instance, in situations where compositions need to be hardened in kilns, and the kiln is sufficiently large that it does not restrict batch sizes. We assume throughout that the jobs and the machine are available from time zero onwards, or equivalently, the jobs have equal release dates. Note that the special case $b = 1$ concurs with the classical single-machine scheduling model in which the machine can handle no more than one job at a time. Accordingly, the bounded model gives rise to problems that are at least as difficult as their traditional counterparts.

For the unbounded model, we give a characterization of a class of optimal schedules, which leads to a generic dynamic programming algorithm for minimizing any regular cost function $\sum_{j=1}^{n} f_j$. This algorithm requires $O(n^2P)$ time and $O(nP)$ space, where P is the sum of the job processing times. The characterization forms the basis of polynomial dynamic programming algorithms for specific cost functions. We present an $O(n^3)$ time algorithm for minimizing the number of tardy jobs $\sum_{j=1}^{n} U_j$, an $O(n \log n)$ time algorithm for minimizing total weighted completion time $\sum_{j=1}^{n} w_j C_j$, and an $O(n^2)$ time algorithm for minimizing the maximum lateness $L_{\max}$. The latter algorithm can be used to construct a polynomial algorithm for minimizing the maximum cost $f_{\max}$. Furthermore, we prove that minimizing the weighted number of tardy jobs $\sum_{j=1}^{n} w_j U_j$ and the total weighted tardiness $\sum_{j=1}^{n} w_j T_j$ are NP-hard problems.

As to the bounded model, Lee *et al.* [1] and Uzsoy [5] present polynomial algorithms for minimizing the number of tardy jobs $\sum_{j=1}^{n} U_j$ and the maximum lateness $L_{\max}$ under a number of assumptions on the relationship between processing times, job release dates, and due dates; see also Reference 6. Minimizing total completion time $\sum_{j=1}^{n} C_j$ is undoubtedly the most vexing bounded problem. Chandru *et al.* [7, 8] present heuristics and a branch-and-bound algorithm as well as an $O(m^3 b^{m+1})$ time dynamic programming algorithm for the case of m different job processing times ($m < n$). Hochbaum and Landy [9] present a faster algorithm that requires $O(m^2 3^m)$ time.

We prove that minimizing total completion time $\sum_{j=1}^{n} C_j$ is solvable in polynomial time for fixed b, where $b > 1$, by deriving an $O(n^{b(b-1)})$ time dynamic programming algorithm for its solution. (The special case where $b = 1$ is a classical scheduling problem that is solvable in $O(n \log n)$ time.) For the case of m different processing times, we give a more efficient dynamic programming algorithm than that of Hochbaum and Landy: ours requires $O(b^2 m^2 2^m)$ time. Furthermore, we prove that other criteria give rise to NP-hard problems. There are two exceptions: minimizing makespan

Table I. Overview of time complexities for problems with equal release dates

Objective function	Unbounded	Bounded	
	$b \geqslant n$	$b = 1$	$b \geqslant 2$
$f_{\max}$	Polynomial	$O(n^2)$	Unary NP-hard
$C_{\max}$	$O(n)$	$O(n)$	$\min\{O(n \log n), O(n^2/b)\}$
$L_{\max}$	$O(n^2)$	$O(n \log n)$	Unary NP-hard
$\sum_{j=1}^{n} f_j$	$O(n^2P)$ Binary NP-hard	Unary NP-hard	Unary NP-hard
$\sum_{j=1}^{n} C_j$	$O(n \log n)$	$O(n \log n)$	$O(n^{b(b-1)})$
$\sum_{j=1}^{n} w_j C_j$	$O(n \log n)$	$O(n \log n)$	Open
$\sum_{j=1}^{n} U_j$	$O(n^3)$	$O(n \log n)$	Unary NP-hard
$\sum_{j=1}^{n} w_j U_j$	$O(n^2P)$ Binary NP-hard	$O(nP)$ Binary NP-hard	Unary NP-hard
$\sum_{j=1}^{n} T_j$	$O(n^2P)$ Open	$O(n^4P)$ Binary NP-hard	Unary NP-hard
$\sum_{j=1}^{n} w_j T_j$	$O(n^2P)$ Binary NP-hard	Unary NP-hard	Unary NP-hard

$C_{\max}$ is solvable in $\min\{O(n \log n), O(n^2/b)\}$ time, while minimizing total completion time $\sum_{j=1}^{n} C_j$ for arbitrary b, and minimizing total weighted completion time $\sum_{j=1}^{n} w_j C_j$ for fixed and arbitrary b are open problems. Finally, we give a polynomial algorithm for minimizing any regular cost function for the special case in which the number of batches is fixed.

Table I summarizes our main results. For sake of comparison, we have included the time complexities of the classical scheduling problems, i.e. problems with $b = 1$; see also Reference 10.

The plan of the rest of this paper is as follows. In Section 2, we give a formal description of the model, introduce our notation, and give some definitions. We analyse the unbounded model in Section 3 and the bounded model in Section 4. In Section 5, we conclude by summarizing our results and point out the consequences for problems with unequal release dates and parallel machines.

2. PROBLEM DESCRIPTION, NOTATION, AND DEFINITIONS

The scheduling model that we analyse is as follows. There are n independent jobs $J_1, \ldots, J_n$ that have to be scheduled on a single batching machine. Each job J_j ($j = 1, \ldots, n$) requires processing during at least a given non-negative uninterrupted time p_j, and is available for processing from time zero onwards. Also, each job J_j has a cost function f_j, where $f_j(t)$ denotes the *cost* incurred if the job is completed at time t. Throughout this paper, we consider only *regular* cost functions, i.e. we assume that $f_j(t)$ is a non-decreasing function of t, for $j = 1, \ldots, n$. Sometimes, each job J_j has a due date d_j by which it should ideally be completed, a deadline $\bar{d}_j$ by which it must be completed, and a weight w_j which is a measure of its importance; when there is ambiguity, we state explicitly whenever due dates, deadlines or weights are present. The weights and due

dates are typically used to define cost objective functions; the deadlines restrict the availability of jobs. The batching machine is available from time zero onwards and can process up to b jobs simultaneously. All jobs in a batch start and complete at the same time. The processing time of a batch is equal to the largest processing time of any job in the batch. Without loss of generality, we assume that the job parameters are integral, unless stated otherwise.

For problems of minimizing a regular objective function without job release dates, we know that there must be at least one optimal solution in which the batches are processed contiguously from time zero onwards. Throughout the paper, we restrict attention to solutions with this property. Thus, a *schedule* σ is a sequence of batches $\sigma = (\mathscr{B}_1, \ldots, \mathscr{B}_r)$, where each batch $\mathscr{B}_l$ $(l = 1, \ldots, r)$ is a set of jobs. The processing time of batch $\mathscr{B}_l$ is $p(\mathscr{B}_l) = \max_{J_j \in \mathscr{B}_l}\{p_j\}$ and its completion time is $C(\mathscr{B}_l) = \sum_{q=1}^{l} p(\mathscr{B}_q)$. Note that the completion time of job J_j in σ, for each $J_j \in \mathscr{B}_l$ and $l = 1, \ldots, r$, is $C_j(\sigma) = C(\mathscr{B}_l)$. When there is no ambiguity, we abbreviate $C_j(\sigma)$ to C_j.

The aim is to minimize the scheduling cost, measured either by a regular minmax objective function $f_{\max} = \max_{1 \leqslant j \leqslant n}\{f_j(C_j)\}$, or by a regular minsum objective function $\sum_{j=1}^{n} f_j = \sum_{j=1}^{n} f_j(C_j)$. Specific regular objective functions that we consider are the makespan $C_{\max}$, defined as $C_{\max} = \max_{1 \leqslant j \leqslant n}\{C_j\}$; maximum lateness $L_{\max}$, defined as $L_{\max} = \max_{1 \leqslant j \leqslant n}\{C_j - d_j\}$; total weighted completion time $\sum_{j=1}^{n} w_j C_j$; total weighted tardiness $\sum_{j=1}^{n} w_j T_j$, where $T_j = \max\{C_j - d_j, 0\}$; and weighted number of tardy jobs $\sum_{j=1}^{n} w_j U_j$, where U_j is 0–1 indicator variable that takes the value 1 if J_j is *tardy*, i.e., if $C_j > d_j$, and the value 0 if J_j is *on time*, i.e., if $C_j \leqslant d_j$. We also provide results for the unweighted versions of these minsum objective functions in which $w_j = 1$ for $j = 1, \ldots, n$.

3. THE UNBOUNDED MODEL

In this section, we assume that $b \geqslant n$ and hence that the batching machine can process any number of jobs at the same time. Note that the problem of minimizing the makespan is solved trivially by putting all jobs in one batch $\mathscr{B}_1$. The minimum makespan is then $C_{\max} = p(\mathscr{B}_1) = \max_{1 \leqslant j \leqslant n}\{p_j\}$.

For the remainder of this section, we assume throughout that the jobs have been re-indexed according to the shortest processing time (SPT) rule so that $p_1 \leqslant \cdots \leqslant p_n$.

3.1. Fundamentals

We first derive a most useful characterization of a class of optimal solutions for minimizing any regular objective function.

Lemma 1. For minimizing any regular objective function, there exists an optimal schedule $(\mathscr{B}_1, \ldots, \mathscr{B}_r)$, *where under the SPT indexing* $\mathscr{B}_l = \{J_{j_l}, J_{j_l+1}, \ldots, J_{j_{l+1}-1}\}$ *and* $1 = j_1 < j_2 < \ldots < j_r < j_{r+1} = n + 1$.

Proof. Consider any optimal schedule $\sigma = (\mathscr{B}_1, \ldots, \mathscr{B}_l, \ldots, \mathscr{B}_q, \ldots, \mathscr{B}_r)$, where $1 \leqslant l < q \leqslant r$, with $J_k \in \mathscr{B}_l$, $J_j \in \mathscr{B}_q$, and $p_k > p_j$. Consider now the schedule $\sigma' = (\mathscr{B}_1, \ldots, \mathscr{B}_l \cup \{J_j\}, \ldots, \mathscr{B}_q \setminus \{J_j\}, \ldots, \mathscr{B}_r)$ that is obtained from σ by moving job J_j to batch $\mathscr{B}_l$. Since $p_j < p_k$, we have that $p(\mathscr{B}_l \cup \{J_j\}) = p(\mathscr{B}_l)$ and $p(\mathscr{B}_q \setminus \{J_j\}) \leqslant p(\mathscr{B}_q)$. Accordingly, the completion time of job J_j decreases from $C(\mathscr{B}_q)$ to $C(\mathscr{B}_l)$, while the completion times of the other jobs do not increase. Since the objective function is regular, the new schedule σ' is also optimal. A finite number of repetitions of this procedure yields an optimal schedule of the required form. ∎

Lemma 1 shows that an optimal schedule is specified by the jobs that start each batch, since the complete schedule can then be formed by the SPT rule. We refer to such a schedule as an *SPT-batch schedule*. The observation that we may restrict our search to SPT-batch schedules forms the basis of a pseudopolynomial dynamic programming algorithm for minimizing any regular minsum objective function $\sum_{j=1}^{n} f_j$, and a polynomial algorithm for minimizing a class of regular minmax objective functions $f_{\max}$.

A dynamic programming algorithm can be built around different types of enumeration schemes. We can use either a forward scheme in which jobs or batches are successively added to the end of the current (partial) schedule, or a backward scheme in which jobs or batches are added to the beginning. The schedule can be constructed by adding either complete batches or single jobs.

We start by explaining in general terms the optimality principle and the working of a generic *forward* dynamic programming algorithm with batch enumeration for a regular minsum objective function $\sum_{j=1}^{n} f_j$ or a regular minmax objective function $f_{\max}$. Consider any feasible SPT-batch schedule containing jobs $J_1,\ldots,J_j$ with the property that the last batch completes at time t. We define such a schedule to be in *state* (j,t). Of course, to schedule the remaining jobs $J_{j+1},\ldots,J_n$, we need to consider only a schedule that has minimum objective value among all schedules in this state.

Let σ be an SPT-batch schedule with minimum objective value from among all schedules in state (j,t). To achieve this state from a previous state, the following decision must be taken to create σ:

- *add a batch containing* J_j. A batch $\{J_{i+1},\ldots,J_j\}$, where $0 \leqslant i < j$, is added to the end of some previous schedule that contains jobs $J_1,\ldots,J_i$. Since batch $\{J_{i+1},\ldots,J_j\}$ has processing time p_j, the previous state is $(i,t-p_j)$. Adding this batch to the previous schedule increases the total cost by $\sum_{k=i+1}^{j} f_k(t)$; the maximum cost of jobs $J_{i+1},\ldots,J_j$ is $\max_{i+1\leqslant k\leqslant j}\{f_k(t)\}$.

This optimality principle leads to a pseudopolynomial $O(n^3 \sum_{j=1}^{n} p_j)$ time dynamic programming algorithm for minimizing any regular objective function, since the state variables are (j,t) for $j=0,\ldots,n$ and $t=0,\ldots,\sum_{i=1}^{n} p_i$, and there are j possible batches that can be added to achieve state (j,t). To implement this algorithm more efficiently for an arbitrary regular minsum objective function, partial sums $\sum_{k=1}^{j} f_k(t)$ are evaluated and stored in a preprocessing step, which reduces the time complexity to $O(n^2 \sum_{j=1}^{n} p_j)$. Details of such an algorithm are given in Section 3.2.

To minimize the number of tardy jobs, we develop a polynomial algorithm by eliminating the state variable t. Since the number of tardy jobs cannot exceed n, we reverse the role of the state variable t and the objective function value. To achieve the $O(n^3)$ time complexity, we build the schedule by adding single jobs instead of complete batches. Full details are presented in Section 3.3.

To construct a polynomial algorithm for other objective functions, we develop a generic *backward* dynamic programming algorithm that allows us to avoid the state variable t. In a backward algorithm, the batches are constructed in the reverse order to which they appear in the schedule. Using this approach, we obtain polynomial algorithms for minimizing the total weighted completion time $\sum_{j=1}^{n} w_j C_j$ and the maximum lateness $L_{\max}$.

We define a cost function f_j to be *additive* if $f_j(t+\Delta) = f_j(t) + f_j(\Delta)$ for any Δ, and to be *incremental* if $f_j(t+\Delta) = f_j(t) + \Delta$ for any Δ. An objective function is additive if each f_j is additive, and is incremental if each f_j is incremental $(j=1,\ldots,n)$. Note that total weighted completion time is additive, and maximum lateness is incremental.

A backward dynamic programming algorithm with batch enumeration for an additive regular minsum objective function $\sum_{j=1}^{n} f_j$ or an incremental regular minmax objective function $f_{\max}$ works as follows. Let σ be an SPT-batch schedule for the jobs $J_j,\ldots,J_n$, where processing of the first batch starts at time zero; we define such a schedule to be in *state* j. To achieve this state from a previous state, the following decision must be taken to create σ:

- *add a batch containing* J_j. A batch $\{J_j,\ldots,J_{k-1}\}$, where $j < k \leqslant n+1$, is inserted at the beginning of some previous schedule in state k. Since batch $\{J_j,\ldots,J_{k-1}\}$ has processing time p_{k-1}, jobs $J_k,\ldots,J_n$ are completed p_{k-1} units later when this batch is added. For an additive objective function, the total cost increases by $\sum_{i=j}^{n} f_i(p_{k-1})$; for an incremental objective function, the maximum cost of jobs $J_k,\ldots,J_n$ increases by p_{k-1}, and the maximum cost of jobs in batch $\{J_j,\ldots,J_{k-1}\}$ is $\max_{j\leqslant i\leqslant k-1}\{f_i(p_{k-1})\}$.

Using a backward recursion, we obtain an $O(n\log n)$ algorithm for minimizing the total weighted completion time in Section 3.4, and an $O(n^2)$ algorithm for minimizing the maximum lateness in Section 3.5.

3.2. Minimizing a regular minsum function

In this section, we formalize the generic forward dynamic programming algorithm with batch enumeration that is outlined in Section 3.1 for the problem of minimizing an arbitrary regular minsum objective function $\sum_{j=1}^{n} f_j$. We show that this problem can be solved in $O(n^2P)$ time and $O(nP)$ space, where $P=\sum_{j=1}^{n} p_j$.

Let $F_j(t)$ be the minimum objective value for SPT-batch schedules containing jobs $J_1,\ldots,J_j$ subject to the condition that the last batch completes at time t. Recall from Section 3.1 that, given $F_j(t)$ and any SPT-batch schedule corresponding to this value, batch $\{J_{i+1},\ldots,J_j\}$, for some i where $0\leqslant i<j$, appears in the last position.

We are now ready to give our dynamic programming recursion. The initialization is

$$F_0(t)=\begin{cases}0 & \text{if } t=0\\ \infty & \text{otherwise}\end{cases}$$

and the recursion for $j=1,\ldots,n$ and $t=p_j,\ldots,\sum_{k=1}^{j} p_k$ is

$$F_j(t)=\min_{0\leqslant i\leqslant j-1}\left\{F_i(t-p_j)+\sum_{k=i+1}^{j} f_k(t)\right\}$$

The optimal solution value is equal to $\min_{p_n\leqslant t\leqslant P}\{F_n(t)\}$, and the corresponding optimal schedule is found by backtracking. To implement the algorithm efficiently, the partial sums $\sum_{k=1}^{j} f_k(t)$ are evaluated and stored for $j=1,\ldots,n$ and $t=p_j,\ldots,\sum_{k=1}^{j} p_k$ in a preprocessing step in $O(nP)$ time. Then, each application of the recursion equation requires $O(n)$ time. Thus, the dynamic algorithm requires $O(n^2P)$ time and $O(nP)$ space.

Woeginger [11] shows that dynamic programming algorithms with a special structure automatically lead to a *fully polynomial approximation scheme*. We note that our dynamic programming algorithm can easily be converted into a form that has this structure.

3.3. Minimizing the number of tardy jobs

In this section, we present an $O(n^3)$ time dynamic programming algorithm for the problem of minimizing the number of tardy jobs. It is a forward algorithm that differs from the generic

pseudopolynomial procedure of Section 3.1 on two counts. First, we use the objective value as a state variable and the makespan as the value of a state; this swap alone is sufficient to develop an $O(n^4)$ algorithm. Second, to obtain an $O(n^3)$ time algorithm, we build the schedule by adding single jobs instead of complete batches and fix the last job to be scheduled in the current batch.

We define a schedule for jobs $J_1,\ldots,J_j$ to be in *state* (j,u,k), where $u \leqslant j \leqslant k$, if it contains exactly u tardy jobs, and the last batch is to be enlarged by including jobs $J_{j+1},\ldots,J_k$, but no others. Thus, a schedule is to be created in which jobs $J_j,\ldots,J_k$ are contained in the same batch, and this batch has processing time p_k. Let $F_j(u,k)$ be the minimum makespan for SPT-batch schedules in state (j,u,k). A schedule in state (j,u,k) with value $F_j(u,k)$, is created by taking one of the following decisions in a previous state:

- *add job J_j so that it does not start the last batch*. The last batch to which J_j is added includes job J_{j-1} and has processing time p_k. This processing time p_k contributes to the makespan of the previous state, which is $F_{j-1}(u,k)$ or $F_{j-1}(u-1,k)$ depending on whether J_j is on time or tardy. If $F_{j-1}(u,k) \leqslant d_j$, then we consider $(j-1,u,k)$ as a previous state with J_j scheduled to be on time; if $F_{j-1}(u-1,k) > d_j$, then we consider $(j-1,u-1,k)$ as a previous state with J_j scheduled to be tardy.
- *add job J_j so that it starts the last batch*. The previous batch ends with job J_{j-1} and the processing time of the new batch is p_k. After adding the contribution from the previous state, the makespan becomes $F_{j-1}(u,j-1)+p_k$ or $F_{j-1}(u-1,j-1)+p_k$, depending on whether J_j is on time or tardy. If $F_{j-1}(u,j-1)+p_k \leqslant d_j$, then we consider $(j-1,u,j-1)$ as a previous state with J_j scheduled to be on time; if $F_{j-1}(u-1,j-1)+p_k > d_j$, then we consider $(j-1,u-1,j-1)$ as a previous state with J_j scheduled to be tardy.

We are now ready to give the dynamic programming recursion. The initialization is

$$F_0(u,k) = \begin{cases} 0 & \text{if } u = 0 \text{ and } k = 0 \\ \infty, & \text{otherwise} \end{cases}$$

and the recursion for $j = 1,\ldots,n$, $u = 0,\ldots,j$, and $k = j,\ldots,n$ is

$$F_j(u,k) = \min \begin{cases} F_{j-1}(u,k) & \text{if } F_{j-1}(u,k) \leqslant d_j \\ F_{j-1}(u-1,k) & \text{if } F_{j-1}(u-1,k) > d_j \\ F_{j-1}(u,j-1)+p_k & \text{if } F_{j-1}(u,j-1)+p_k \leqslant d_j \\ F_{j-1}(u-1,j-1)+p_k & \text{if } F_{j-1}(u-1,j-1)+p_k > d_j \\ \infty & \text{otherwise} \end{cases}$$

The minimum number of tardy jobs is then equal to the smallest value u for which $F_n(u,n) < \infty$, and the corresponding optimal schedule is found by backtracking. Note that the algorithm requires $O(n^3)$ time and $O(n^3)$ space.

3.4. Minimizing total weighted completion time

In this section, we present an $O(n \log n)$ time dynamic programming algorithm for minimizing the total weighted completion time $\sum_{j=1}^{n} w_j C_j$. Since we have an additive minsum objective function, the generic backward dynamic programming algorithm of Section 3.1 can be used.

Let F_j be the minimum total weighted completion time for SPT-batch schedules containing the *last* $n-j+1$ jobs $J_j,\ldots,J_n$. Processing of the first batch in the schedule starts at time zero. Furthermore, whenever a new batch is added to the beginning of this schedule, there is a corresponding delay in the processing of all batches. Suppose that a batch $\{J_j,\ldots,J_{k-1}\}$, which has

processing time p_{k-1}, is inserted at the start of a schedule for jobs $J_k,\ldots,J_n$. The total weighted completion time of jobs $J_k,\ldots,J_n$ increases by $p_{k-1}\sum_{i=k}^{n} w_i$, while the total weighted completion time for jobs $J_j,\ldots,J_{k-1}$ is $p_{k-1}\sum_{i=j}^{k-1} w_i$. Thus, the overall increase in total weighted completion time is $p_{k-1}\sum_{i=j}^{n} w_i$.

We are now ready to give the dynamic programming recursion. The initialization is

$$F_{n+1} = 0$$

and the recursion for $j = n, n-1,\ldots,1$ is

$$F_j = \min_{j<k\leqslant n+1}\left\{F_k + p_{k-1}\sum_{i=j}^{n} w_i\right\}$$

The optimal solution value is then equal to F_1, and the corresponding optimal schedule is found by backtracking. Under the most natural implementation, the algorithm requires $O(n^2)$ time and $O(n)$ space, if we compute and store the values $\sum_{i=j}^{n} w_i$ for $j = 1,\ldots,n$ in a preprocessing step. However, since our dynamic program has a structure that allows geometric techniques to be applied [12], the time complexity can be reduced to $O(n\log n)$.

3.5. Minimizing maximum lateness and maximum cost

In this section, we present an $O(n^2)$ dynamic programming algorithm for minimizing the maximum lateness $L_{\max}$. This algorithm serves as a subroutine for a polynomial algorithm that minimizes the maximum cost $f_{\max}$. Since $L_{\max}$ is an incremental minmax objective function, we employ a backward recursion of the type given in Section 3.1.

Let F_j be the minimum value of the maximum lateness for SPT-batch schedules containing the *last* $n-j+1$ jobs $J_j,\ldots,J_n$, where processing starts at time zero. If a batch $\{J_j,\ldots,J_{k-1}\}$, which has processing time p_{k-1}, is inserted at the start of a schedule for jobs $J_k,\ldots,J_n$, then the maximum lateness of jobs $J_k,\ldots,J_n$ increases by p_{k-1}, while the maximum lateness for jobs $J_j,\ldots,J_{k-1}$ is $\max_{j\leqslant i\leqslant k-1}\{p_{k-1}-d_i\}$.

We are now ready to give the dynamic programming recursion. The initialization is

$$F_{n+1} = -\infty$$

and the recursion for $j = n, n-1,\ldots,1$ is

$$F_j = \min_{j<k\leqslant n+1}\left\{\max\{F_k + p_{k-1}, \max_{j\leqslant i\leqslant k-1}\{p_{k-1}-d_i\}\}\right\}$$

The optimal solution value is then equal to F_1, and the corresponding optimal schedule is found by backtracking. Note that the algorithm requires $O(n^2)$ time and $O(n)$ space.

We now show how to construct a polynomial algorithm for minimizing $f_{\max}$ using the $O(n^2)$ algorithm for minimizing $L_{\max}$ as a subroutine. First, note that the problem of minimizing $f_{\max}$ can be viewed as a finite series of decision problems of the type 'is $f_{\max}\leqslant k$?', where k is repeatedly adjusted by binary search over an appropriate interval for k. Hence, if the decision problem is solvable in polynomial time, then minimizing $f_{\max}$ is solvable in polynomial time if the optimal solution value is an integer whose logarithm is polynomially bounded in the size of the input. We assume that this is so. This assumption is not very restrictive since it holds when f_j has the form $f_j = (\beta_j)^{\alpha_j}$, where $\beta_j \in \{w_jC_j, w_jT_j, w_jU_j\}$, and α_j is a polynomial in n. The question 'is $f_{\max}\leqslant k$?' can be answered in polynomial time as follows. Observe that the upper bound k induces a deadline $\bar{d}_j$ on the completion time of each job J_j, for $j = 1,\ldots,n$. Each deadline can

be determined in $O(\log P)$ time by binary search over the $P+1$ possible completion times. Once the deadlines have been determined, we can use the algorithm for minimizing $L_{\max}$ to find out if there is a solution in which each job is completed before its deadline by treating the deadlines as due dates: if $L_{\max} \leqslant 0$, then a schedule exists in which no deadlines are violated; otherwise, no such schedule exists. Hence, the question 'is $f_{\max} \leqslant k$?' can be answered in $O(n^2 + n \log P)$ time, which is polynomial, and the problem of minimizing $f_{\max}$ can be solved in polynomial time.

3.6. Minimizing the weighted number of tardy jobs

In Section 3.2, we establish that problems of minimizing a regular minsum function $\sum_{j=1}^{n} f_j$, such as the weighted number of tardy jobs, can be solved by a pseudopolynomial dynamic programming algorithm in $O(n^2 P)$ time, and that there is a fully polynomial approximation scheme. Also, the algorithm in Section 3.3 can be generalized to minimize the weighted number of tardy jobs in $O(n^2 \sum_{j=1}^{n} w_j)$ time. In this section, we show that there exists no polynomial algorithm for this problem, unless P = NP: we prove here that this problem is NP-hard in the ordinary sense.

Theorem 1. The unbounded problem of minimizing the weighted number of tardy jobs $\sum_{j=1}^{n} w_j U_j$ is binary NP-hard.

Proof. Our proof proceeds by a reduction from the binary NP-complete problem PARTITION.

PARTITION

Given a set $\{a_1, \ldots, a_m\}$ of m positive integers, is it possible to partition the index set $\{1, \ldots, m\}$ into two disjoint subsets $\mathscr{X}$ and $\mathscr{Y}$ such that $\sum_{j \in \mathscr{X}} a_j = A$, where $A = \sum_{j=1}^{m} a_j/2$?

Given an instance of PARTITION, we define an instance of the unbounded weighted number of tardy jobs problem with $n = 2m$ jobs and $b = n$. For each j ($j = 1, \ldots, m$), we define a 'light' job J_j with $p_j = 2jA + a_j$, $w_j = a_j$, and $d_j = (j^2 + j + 1)A$, and a 'heavy' job J_{m+j} with $p_{m+j} = 2jA$, $w_{m+j} = A + 1$, and $d_{m+j} = (j^2 + j + 1)A$. Note that J_j and J_{m+j} have the same due date, for $j = 1, \ldots, m$.

We first show that the reduction for constructing the instance of the unbounded weighted number of tardy jobs is polynomial. Under a binary encoding, obvious lower bounds on the size of the input for PARTITION are m and $\log_2 A$. For the unbounded weighted number of tardy jobs problem, each processing time is bounded above by $(2m + 1)A$. An upper bound on the total size of the processing times under a binary encoding is $2m \log_2(2m + 1) + 2m \log_2 A$, which is polynomially bounded in m and $\log_2 A$. Similarly, upper bounds on the total size of the weights and due dates under a binary encoding are $2m \log_2(A + 1)$ and $2m \log_2(m^2 + m + 1) + 2m \log_2 A$, respectively, which are also polynomially bounded in m and $\log_2 A$. Thus, our reduction is polynomial.

In the remainder of the proof, we show that PARTITION has a solution if and only if there exists a schedule for the corresponding instance of the weighted number of tardy jobs problem with $\sum_{j=1}^{n} w_j U_j \leqslant A$.

First, suppose that $\mathscr{X}$ and $\mathscr{Y}$ define a solution to PARTITION. Consider a schedule with $m + 1$ batches that is constructed as follows. Each 'light' job J_j, for $j = 1, \ldots, m$, is assigned to batch $\mathscr{B}_j$ if $j \in \mathscr{X}$, and is assigned to batch $\mathscr{B}_{m+1}$ if $j \in \mathscr{Y}$. The 'heavy' jobs $J_{m+1}, \ldots, J_{2m}$ are assigned to batches $\mathscr{B}_1, \ldots, \mathscr{B}_m$, respectively. The processing time of batch $\mathscr{B}_j$, for $j = 1, \ldots, m$, is either $2jA + a_j$ or $2jA$ depending on whether or not $j \in \mathscr{X}$. Since $C(\mathscr{B}_j) \leqslant \sum_{i=1}^{j} 2iA + \sum_{i \in \mathscr{X}} a_i = d_j =$

d_{m+j} for $j = 1,\ldots,m$, each 'heavy' job and each 'light' job J_j for $j \in \mathscr{X}$ are on time. Therefore, $\sum_{j=1}^{n} w_j U_j \leqslant \sum_{j\in\mathscr{Y}} w_j = A$.

Conversely, suppose that there exists a schedule with $\sum_{j=1}^{n} w_j U_j \leqslant A$. In such a schedule, all 'heavy' jobs have to be on time. Hence, J_{m+1} has to be processed in batch $\mathscr{B}_1$, and neither job J_j nor J_{m+j} with $j > 1$ can be processed together with it in $\mathscr{B}_1$. Accordingly, J_{m+2} is processed in batch $\mathscr{B}_2$, which cannot begin before time $2A$, the earliest possible completion time of $\mathscr{B}_1$. Since J_{m+2} has to be completed by its due date $7A$, neither job J_j nor J_{m+j} with $j > 2$ can be processed together with it in $\mathscr{B}_2$. Further, the earliest completion time of $\mathscr{B}_2$ is $6A$, so job J_1 is tardy if it is processed in $\mathscr{B}_2$. Repeating this line of reasoning, we deduce that each 'heavy' job J_{m+j} and each on-time 'light' job J_j are assigned to batch $\mathscr{B}_j$, for $j = 1,\ldots,m$. Moreover, we assume without loss of generality that each tardy 'light' job is assigned to batch $\mathscr{B}_{m+1}$.

If job J_j is assigned to batch $\mathscr{B}_j$, then $p(\mathscr{B}_j) = 2jA + a_j$; otherwise, $p(\mathscr{B}_j) = 2jA$. Let $\mathscr{X}$ and $\mathscr{Y}$ denote the set of indices j ($j = 1,\ldots,m$) for which $J_j \in \mathscr{B}_j$ and $J_j \notin \mathscr{B}_j$, respectively. To ensure that job J_{2m} is on time, we require that $C(\mathscr{B}_m) = \sum_{j=1}^{m} 2jA + \sum_{j\in\mathscr{X}} a_j \leqslant (m^2 + m + 1)A$. Thus, $\sum_{j\in\mathscr{X}} a_j \leqslant A$. The condition $\sum_{j=1}^{n} w_j U_j \leqslant A$ implies that $\sum_{j\in\mathscr{Y}} a_j \leqslant A$, or equivalently $\sum_{j\in\mathscr{X}} a_j \geqslant A$. Therefore, $\sum_{j\in\mathscr{X}} a_j = A$, which shows that $\mathscr{X}$ and $\mathscr{Y}$ define a solution to PARTITION. ■

3.7. Minimizing total weighted tardiness

For minimizing the total weighted tardiness, we obtain a similar result to that for minimizing the weighted number of tardy jobs. This problem is solvable in pseudopolynomial time $O(n^2P)$ and that there is a fully polynomial approximation scheme, which is shown in Section 3.2, and is NP-hard in the ordinary sense, which we prove here. Hence, a pseudopolynomial algorithm for its solution is the best we can achieve, unless P = NP.

Theorem 2. The unbounded problem of minimizing the total weighted tardiness $\sum_{j=1}^{n} w_j T_j$ is binary NP-hard.

Proof. Our proof proceeds again by a reduction from PARTITION; see Section 3.6. For (our) convenience, we describe the construction using fractional weights. By multiplying all weights with a suitable number, however, we can obtain a proof with integral parameters only.

Given an instance of PARTITION, we construct an instance of the unbounded total weighted tardiness problem with $n = 2m$ jobs and $b = n$. For each j ($j = 1,\ldots,m$), we define a 'light' job J_j with $p_j = 2jmA^2 + a_j$, $w_j = a_j/(2(j+1)mA^2)$, and $d_j = A + j(j+1)mA^2$, and a 'heavy' job J_{m+j} with $p_{m+j} = 2jmA^2$, $w_{m+j} = A + 1$, and $d_{m+j} = A + j(j+1)mA^2$. Note that J_j and J_{m+j} have the same due date, for $j = 1,\ldots,m$.

We first show that the reduction for constructing the instance of the unbounded total weighted tardiness problem is polynomial. As in the proof of Theorem 1, lower bounds on the size of the input for PARTITION under a binary encoding are m and $\log_2 A$. For the unbounded total weighted tardiness problem, upper bounds on the total size of the processing times, the numerators of the weights, the denominators for the weights, and the due dates under a binary encoding are $2m\log_2(2m^2A^2 + A)$, $2m\log_2 A$, $2m\log_2(2m(m+1)A^2)$, and $2m\log_2(m^2(m+1)A^2 + A)$, respectively, which are polynomially bounded in m and $\log_2 A$. Thus, our reduction is polynomial.

In the remainder of the proof, we show that PARTITION has a solution if and only if there is a schedule for the corresponding instance of the total weighted tardiness problem with $\sum_{j=1}^{n} w_j T_j \leqslant A$.

First, suppose that $\mathscr{X}$ and $\mathscr{Y}$ define a solution to PARTITION. We assume without loss of generality that $m \in \mathscr{X}$. Consider a schedule with m batches that is constructed as follows. Each 'light' job J_j, for $j = 1,\ldots,m$, is assigned to batch $\mathscr{B}_j$ if $j \in \mathscr{X}$, and is assigned to batch $\mathscr{B}_{j+1}$ if $j \in \mathscr{Y}$. The 'heavy' jobs $J_{m+1},\ldots,J_{2m}$ are assigned to batches $\mathscr{B}_1,\ldots,\mathscr{B}_m$ respectively. The processing time of batch $\mathscr{B}_j$, for $j = 1,\ldots,m$, is either $2jmA^2 + a_j$ or $2jmA^2$ depending on whether or not $j \in \mathscr{X}$. Since $C(\mathscr{B}_j) \leqslant \sum_{i=1}^{j} 2imA^2 + \sum_{i\in\mathscr{X}} a_i = d_j = d_{m+j}$ for $j = 1,\ldots,m$, each 'heavy' job and each 'light' job J_j for $j \in \mathscr{X}$ are on time. Moreover, each 'light' job J_j for $j \in \mathscr{Y}$ is completed at time $C_j = C(\mathscr{B}_{j+1})$, where $C(\mathscr{B}_{j+1}) \geqslant \sum_{i=1}^{j+1} 2imA^2 > d_j$ and is therefore tardy. Further, $C_j \leq \sum_{i=1}^{j+1} 2imA^2 + \sum_{i\in\mathscr{X}} a_i = d_{j+1}$ for $j \in \mathscr{Y}$. Therefore, $\sum_{j=1}^{n} w_j T_j \leqslant \sum_{j\in\mathscr{Y}} w_j(d_{j+1} - d_j) = \sum_{j\in\mathscr{Y}} w_j(2j+2)mA^2 = \sum_{j\in\mathscr{Y}} a_j = A$.

Conversely, assume that there exists a schedule with $\sum_{j=1}^{n} w_j T_j \leqslant A$. In any such schedule, all 'heavy' jobs have to be on time. Also, as in the proof of Theorem 1, it is straightforward to show that in any such schedule each 'heavy' job J_{m+j} and each on-time 'light' job J_j are assigned to batch $\mathscr{B}_j$, and each tardy 'light' job is assigned to one of the batches $\mathscr{B}_{j+1},\ldots,\mathscr{B}_{m+1}$, where $\mathscr{B}_{m+1}$ is a final batch that contains only tardy jobs.

If job J_j is assigned to batch $\mathscr{B}_j$, then $p(\mathscr{B}_j) = 2jmA^2 + a_j$; otherwise, $p(\mathscr{B}_j) = 2jmA^2$. Let $\mathscr{X}$ and $\mathscr{Y}$ denote the set of indices j $(j = 1,\ldots,m)$ for which $J_j \in \mathscr{B}_j$ and $J_j \notin \mathscr{B}_j$, respectively. To ensure that job J_{2m} is on time, we require that $C(\mathscr{B}_m) = \sum_{j=1}^{m} 2jmA^2 + \sum_{j\in\mathscr{X}} a_j \leqslant A + m^2(m+1)A^2$. Thus, $\sum_{j\in\mathscr{X}} a_j \leqslant A$.

Each tardy 'light' job J_j, where $j \in \mathscr{Y}$, is assigned to one of the batches $\mathscr{B}_{j+1},\ldots,\mathscr{B}_{m+1}$. Hence, for $j \in \mathscr{Y}$, we have $C_j \geqslant \sum_{i=1}^{j+1} 2imA^2$, $T_j \geqslant 2(j+1)mA^2 - A$ and $w_j T_j \geqslant a_j - a_j/((2j+2)mA) > a_j - 1/(2m)$. Therefore, $\sum_{j=1}^{n} w_j T_j > \sum_{j\in\mathscr{Y}}(a_j - 1/(2m)) > \sum_{j\in\mathscr{Y}} a_j - 1$. Since the a_j's are integral and $\sum_{j=1}^{n} w_j T_j \leqslant A$, we must also have $\sum_{j\in\mathscr{Y}} a_j \leqslant A$, or equivalently $\sum_{j\in\mathscr{X}} a_j \geqslant A$. Therefore, $\sum_{j\in\mathscr{X}} a_j = A$, which shows that $\mathscr{X}$ and $\mathscr{Y}$ define a solution to PARTITION. ■

4. THE BOUNDED MODEL

In this section, we analyse problems in which b is restrictively small: we assume that $b < n$. These *bounded problems* are at least as difficult as their traditional counterparts, since for the special case $b = 1$ the machine can handle no more than one job at a time. They are also inherently much more difficult than their unbounded counterparts, mainly because we can no longer restrict ourselves to SPT-batch schedules in the search for an optimal schedule. There is one exception: for minimizing makespan, there is still an SPT-batch schedule that is optimal.

For the bounded problem of minimizing the makespan, we assume that n is an integer multiple of b and that $n = br$. This assumption is justified by the observation that dummy jobs with zero processing time can be introduced without affecting the minimum makespan. The problem is solved by assigning the b jobs with smallest processing times to $\mathscr{B}_1$, the b jobs with the next smallest processing times to $\mathscr{B}_2$, and so on, until the b jobs with largest processing times are assigned to $\mathscr{B}_r$. Hence, we can solve the problem in $O(n \log n)$ time, if we first order the jobs using the SPT rule. Alternatively, we can solve the problem in $O(rn)$ time, if we use linear time median finding techniques [13, 14]. More precisely, we can determine $\mathscr{B}_1 \cup \cdots \cup \mathscr{B}_l$, for $l = 1,\ldots,r$, by finding some job J_j in $O(n)$ time such that $|\{i|\, p_i < p_j\}| < bl$ and $|\{i|\, p_i > p_j\}| < b(r-l)$ and then introducing a subset of $\{i|\, p_i = p_j\}$ into the first set so that its cardinality is exactly equal to bl. Under this implementation, the problem is solved in $O(n^2/b)$ time.

In Section 4.1, we analyse the problem of minimizing total completion time. We show that the problem can be solved in $O(n^{b(b-1)})$ time by dynamic programming for $b > 1$. Furthermore, we give an $O(b^2m^2 2^m)$ time algorithm for the case of m different processing times.

We also provide complexity results for scheduling with due dates. Specifically, we show in Section 4.2 that finding whether there is a feasible solution to the bounded problem in which the jobs have deadlines is NP-complete in the strong sense, even if $b = 2$. This result implies that minimizing the maximum lateness, the number of tardy jobs, and the total tardiness are all unary NP-hard problems.

Finally, Section 4.3 addresses the special case in which the number of batches to be used is fixed. We show that bounded problems of this type can be solved in $O(n^{r+3})$ time, where r is the given number of batches.

4.1. Minimizing total completion time

The bounded problem of minimizing total completion time is introduced by Chandru *et al.* [7], who present a branch-and-bound algorithm and some heuristics. We show in Section 4.1.1 that the general problem is solvable in $O(n^{b(b-1)})$ time and $O(n^{b(b-1)})$ space for $b > 1$. For the case of m different job types (where $m < n$), Chandru *et al.* [8] present an $O(m^3 b^{m+1})$ time dynamic programming algorithm. Hochbaum and Landy [9] present a more efficient algorithm that requires $O(m^2 3^m)$ time. In Section 4.1.2, we present an algorithm with a further gain in efficiency: our algorithm requires $O(b^2m^2 2^m)$ time and $O(bm^2)$ space.

Throughout this section, we again assume that the jobs have been re-indexed according to the SPT rule so that $p_1 \leqslant \cdots \leqslant p_n$. We now present two results of Chandru *et al.* [7]. The first result shows that there exists an optimal schedule in which each batch contains jobs with consecutive indices.

Lemma 2 (Chandru *et al.* [7]). *There exists an optimal schedule* $(\mathscr{B}_1, \ldots, \mathscr{B}_r)$ *for which, under the SPT indexing,* $\mathscr{B}_l = \{J_{i_l}, \ldots, J_{j_l}\}$, *where* $1 \leqslant i_l \leqslant j_l \leqslant n$, *for* $l = 1, \ldots, r$.

The second result of Chandru *et al.* [7] extends the classical SWPT rule for sequencing jobs on a single machine to the sequencing of batches.

Lemma 3 (Chandru *et al.* [7]). *For given batches* $\mathscr{B}_1, \ldots, \mathscr{B}_r$, *an optimal sequence is* $(\mathscr{B}_1, \ldots, \mathscr{B}_r)$ *if and only if*

$$p(\mathscr{B}_1)/|\mathscr{B}_1| \leqslant \cdots \leqslant p(\mathscr{B}_r)/|\mathscr{B}_r| \tag{1}$$

We define a batch to be *full* if it contains exactly b jobs; otherwise, it is *non-full*. Also, a batch $\mathscr{B}_l$ is *deferred* with respect to another batch $\mathscr{B}_q$ if $\mathscr{B}_l$ is sequenced after $\mathscr{B}_q$ and $p(\mathscr{B}_l) < p(\mathscr{B}_q)$.

We now present a result of Hochbaum and Landy [9] relating to deferred batches.

Lemma 4 (Hochbaum and Landy [9]). *In any optimal schedule, there is no batch that is deferred with respect to a non-full batch.*

In our subsequent analysis, we only consider schedules that are consistent with the above lemmas: each batch contains jobs with consecutive indices; batches are ordered according to (1); and no batch is deferred with respect to a non-full batch.

4.1.1. A polynomial algorithm for fixed b

In this subsection, we derive a polynomial algorithm for the bounded problem of minimizing the total completion time. The special case $b = 1$ is equivalent to the corresponding classical scheduling problem for which the SPT rule provides an optimal solution in $O(n \log n)$ time. Thus, we assume that $b \geqslant 2$, and derive an $O(n^{b(b-1)})$ dynamic programming algorithm. This algorithm relies on an upper bound of the number of deferred batches, which we establish next.

Lemma 5. In any optimal schedule, the number of deferred batches with respect to any full batch does not exceed $b^2 - b - 1$.

Proof. Consider any schedule σ for which the number of deferred batches with respect to some full batch $\mathscr{B}_l$ is at least $b^2 - b$. We show that σ cannot be an optimal schedule.

We first note from Lemma 3 that all full batches are sequenced in non-decreasing order of their processing times: consequently, any deferred batch is non-full. Suppose that the deferred batches with respect to batch $\mathscr{B}_l$ in σ are divided into two groups so that the first group comprises the first $(b-1)^2 + 1$ of these batches and the second group comprises the remainder of these deferred batches. We observe that, in the first group, there are at least b deferred batches containing the same number of jobs; let a denote the number of jobs in each of these batches, where $a < b$.

It is useful to represent σ in the form

$$\sigma = (\mathscr{S}_0, \mathscr{A}_1, \mathscr{S}_1, \mathscr{A}_2, \ldots, \mathscr{S}_{b-1}, \mathscr{A}_b, \mathscr{S}_b)$$

where $\mathscr{A}_i$, for $i = 1, \ldots, b$, is a batch that contains a jobs and is deferred with respect to batch $\mathscr{B}_l$, and where $\mathscr{S}_i$, for $i = 0, \ldots, b$, is a block of batches. Note that block $\mathscr{S}_0$ contains batch $\mathscr{B}_l$ and block $\mathscr{S}_b$ contains all deferred batches of the second group. Moreover, $p(\mathscr{A}_1) \leqslant \cdots \leqslant p(\mathscr{A}_b)$ from Lemma 3. Let $p(\mathscr{S}_i)$ and $|\mathscr{S}_i|$ denote the total processing time of the batches in block $\mathscr{S}_i$ and the number of jobs in block $\mathscr{S}_i$, respectively, for $i = 1, \ldots, b$. Since the number of deferred batches with respect to batch $\mathscr{B}_l$ is at least $b^2 - b$, there are at least $b^2 - b - (b-1)^2 - 1 = b - 2$ batches in $\mathscr{S}_b$, and consequently $|\mathscr{S}_b| \geqslant b - 2$.

Suppose that we construct full batches $\mathscr{A}'_{b-a+1}, \ldots, \mathscr{A}'_b$ from batches $\mathscr{A}_{b-a+1}, \ldots, \mathscr{A}_b$ by adding the a jobs from each of the batches $\mathscr{A}_1, \ldots, \mathscr{A}_{b-a}$. For example, the a respective jobs of batch $\mathscr{A}_i$, could be added to batches $\mathscr{A}_{b-a+1}, \ldots, \mathscr{A}_b$, for $i = 1, \ldots, b-a$. We now define the schedule

$$\sigma_1 = (\mathscr{S}_0, \mathscr{A}'_{b-a+1}, \ldots, \mathscr{A}'_b, \mathscr{S}_1, \ldots, \mathscr{S}_b)$$

Our analysis also uses two artificial schedules σ' and σ'_1 for related problems in which some batches are removed and replaced by duplicates of other batches in the schedule. Specifically, schedule σ' is obtained from σ by the following transformation. First, we replace each batch $\mathscr{A}_i$ for $i = 1, \ldots, b-a$ with $\mathscr{A}_1$, and each batch $\mathscr{A}_i$ for $i = b-a+1, \ldots, b$, with $\mathscr{A}_b$. Second, we sequence the $b-a$ batches of type $\mathscr{A}_1$ in adjacent positions, and also sequence the a batches of type $\mathscr{A}_b$ in adjacent positions, with blocks $\mathscr{S}_1, \ldots, \mathscr{S}_{b-1}$ in between. Thus,

$$\sigma' = (\mathscr{S}_0, \underbrace{\mathscr{A}_1, \ldots, \mathscr{A}_1}_{b-a \text{ batches}}, \mathscr{S}_1, \mathscr{S}_2, \ldots, \mathscr{S}_{b-1}, \underbrace{\mathscr{A}_b, \ldots, \mathscr{A}_b}_{a \text{ batches}}, \mathscr{S}_b)$$

Schedule σ'_1 is constructed from σ_1 by replacing each batch $\mathscr{A}'_i$, for $i = b-a+1, \ldots, b$, with $\mathscr{A}'_b$: hence,

$$\sigma'_1 = (\mathscr{S}_0, \underbrace{\mathscr{A}'_b, \ldots, \mathscr{A}'_b}_{a \text{ batches}}, \mathscr{S}_1, \mathscr{S}_2, \ldots, \mathscr{S}_{b-1}, \mathscr{S}_b)$$

Assume that σ is an optimal schedule. We prove the lemma by deriving two contradictory inequalities relating $\sum_{j=1}^{n} C_j(\sigma')$ and $\sum_{j=1}^{n} C_j(\sigma_1')$. First, we establish that

$$\sum_{j=1}^{n} C_j(\sigma') \leqslant \sum_{j=1}^{n} C_j(\sigma_1') \tag{2}$$

Comparing schedules σ_1' and σ_1, we observe that there is an increase in the processing time of $p(\mathscr{A}_b') - p(\mathscr{A}_i')$ when batch $\mathscr{A}_i'$ is substituted by $\mathscr{A}_b'$, for $i = b-a+1, \ldots, b$, and this increase also delays all subsequent jobs in σ_1'. From the construction of $\mathscr{A}_i'$ from $\mathscr{A}_i$, for $i = b-a+1, \ldots, b$, we observe that $p(\mathscr{A}_i') = p(\mathscr{A}_i)$. Thus, we deduce that

$$\sum_{j=1}^{n} C_j(\sigma_1') = \sum_{j=1}^{n} C_j(\sigma_1) + \sum_{i=b-a+1}^{b} (p(\mathscr{A}_b) - p(\mathscr{A}_i)) \left((b-i+1)b + \sum_{l=1}^{b} |\mathscr{S}_l| \right) \tag{3}$$

We now compare schedules σ' and σ. First, observe that replacing batch $\mathscr{A}_i$, for $i = 1, \ldots, b-a$, with $\mathscr{A}_1$ does not increase the total completion time since $p(\mathscr{A}_1) \leqslant p(\mathscr{A}_i)$ from Lemma 3. Moreover, Lemma 3 also shows that reordering these $\mathscr{A}_1$ batches so that they are sequenced in adjacent positions does not increase the total completion time either. To obtain an upper bound on the increase in total completion time when batch $\mathscr{A}_i$ is substituted by $\mathscr{A}_b$, for $i = b-a+1, \ldots, b$, we assume that the increase in processing time of $p(\mathscr{A}_b) - p(\mathscr{A}_i)$ causes a delay to each of the blocks $\mathscr{S}_1, \ldots, \mathscr{S}_b$. Thus, we obtain

$$\sum_{j=1}^{n} C_j(\sigma') \leqslant \sum_{j=1}^{n} C_j(\sigma) + \sum_{i=b-a+1}^{b} (p(\mathscr{A}_b) - p(\mathscr{A}_i)) \left((b-i+1)b + \sum_{l=1}^{b} |\mathscr{S}_l| \right) \tag{4}$$

Subtracting (3) from (4) yields

$$\sum_{j=1}^{n} C_j(\sigma') - \sum_{j=1}^{n} C_j(\sigma_1') \leqslant \sum_{j=1}^{n} C_j(\sigma) - \sum_{j=1}^{n} C_j(\sigma_1) \leqslant 0$$

where the latter inequality holds due to the optimality of σ. Therefore, we have established the desired inequality (2).

To obtain the required contradiction, it is sufficient to prove that

$$\sum_{j=1}^{n} C_j(\sigma') > \sum_{j=1}^{n} C_j(\sigma_1') \tag{5}$$

Let K denote the total completion time of jobs in the schedule defined by $(\mathscr{S}_0, \ldots, \mathscr{S}_b)$. In schedule σ_1', each of the blocks $\mathscr{S}_1, \ldots, \mathscr{S}_b$ is delayed by time $ap(\mathscr{A}_b')$. Since $p(\mathscr{A}_b') = p(\mathscr{A}_b)$, we deduce that

$$\sum_{j=1}^{n} C_j(\sigma_1') = K + ab\,p(\mathscr{S}_0) + ab(a+1)\,p(\mathscr{A}_b)/2 + a\,p(\mathscr{A}_b) \sum_{l=1}^{b} |\mathscr{S}_l| \tag{6}$$

Performing similar calculations for schedule σ', we obtain

$$\begin{aligned} \sum_{j=1}^{n} C_j(\sigma') = K &+ a(b-a)\,p(\mathscr{S}_0) + a(b-a)(b-a+1)\,p(\mathscr{A}_1)/2 + (b-a)\,p(\mathscr{A}_1) \sum_{l=1}^{b} |\mathscr{S}_l| \\ &+ a^2 \left(\sum_{l=0}^{b-1} p(\mathscr{S}_l) + (b-a)\,p(\mathscr{A}_1) \right) + a^2(a+1)\,p(\mathscr{A}_b)/2 + a\,p(\mathscr{A}_b)|\mathscr{S}_b| \end{aligned} \tag{7}$$

Subtracting (6) from (7) yields

$$\sum_{j=1}^{n} C_j(\sigma') - \sum_{j=1}^{n} C_j(\sigma'_1) = (b-a)\,p(\mathscr{A}_1)\left(a(b+a+1)/2 + \sum_{l=1}^{b} |\mathscr{S}_l|\right) - a\,p(\mathscr{A}_b)\left((a+1)(b-a)/2 + \sum_{l=1}^{b-1} |\mathscr{S}_l|\right) + a^2 \sum_{l=1}^{b-1} p(\mathscr{S}_l) \tag{8}$$

Rearranging (8), we obtain

$$\sum_{j=1}^{n} C_j(\sigma') - \sum_{j=1}^{n} C_j(\sigma'_1) = (b-a)\,p(\mathscr{A}_1)((a^2+a-b)/2 + |\mathscr{S}_b|)) + (bp(\mathscr{A}_1) - ap(\mathscr{A}_b))\left((a+1)(b-a)/2 + \sum_{l=1}^{b-1} |\mathscr{S}_l|\right) + a\left(a\sum_{l=1}^{b-1} p(\mathscr{S}_l) - p(\mathscr{A}_1)\sum_{l=1}^{b-1} |\mathscr{S}_l|\right) \tag{9}$$

Using our previous observation that $|\mathscr{S}_b| \geqslant b-2$, together with $b \geqslant 2$ and $a \geqslant 1$, shows that $(a^2 + a - b)/2 + |\mathscr{S}_b| \geqslant 0$. Thus, the first term in equation (9) is non-negative. Applying Lemma 3 to the batches $\mathscr{B}_l$ and $\mathscr{A}_1$ in σ, we obtain $p(\mathscr{B}_l)/b \leqslant p(\mathscr{A}_1)/a$. Since batch $\mathscr{A}_b$ is deferred with respect to batch $\mathscr{B}_l$, we have that $p(\mathscr{A}_b) < p(\mathscr{B}_l)$. Combining these inequalities yields $p(\mathscr{A}_b)/b < p(\mathscr{A}_1)/a$. Thus, the second term in equation (9) is strictly positive. A further application of Lemma 3 to batch $\mathscr{A}_1$ and to all of the batches in blocks $\mathscr{S}_1, \ldots, \mathscr{S}_{b-1}$ in σ yields $p(\mathscr{A}_1)/a \leqslant \sum_{l=1}^{b-1} p(\mathscr{S}_l) / \sum_{l=1}^{b-1} |\mathscr{S}_l|$. This establishes that the third term in equation (9) is non-negative.

We have now proved that inequality (5) holds, which contradicts (2). Therefore, σ is not an optimal schedule. This implies that the number of deferred batches with respect to any full batch in an optimal schedule is at most $b^2 - b - 1$. ■

We now present a backward dynamic programming algorithm that uses state variables to identify deferred batches. Lemma 5 establishes an upper bound on the number of state variables of this type. Recall our assumption that the jobs have been re-indexed according to the Shortest Processing Time (SPT) rule. We also assume that all processing times are distinct, which can be achieved if necessary by perturbation.

Let σ be a schedule that contains jobs $J_j, \ldots, J_n$, but not J_{j-1}, and also contains non-full batches $\mathscr{B}_1, \ldots, \mathscr{B}_r$, where $p(\mathscr{B}_1) \leqslant \cdots \leqslant p(\mathscr{B}_r)$, which are deferred with respect to the batch containing job J_{j-1} that remains to be scheduled. Processing of the first batch in σ starts at time zero. From Lemma 2, we may assume that $\mathscr{B}_l = \{J_{j_l}, \ldots, J_{j'_l}\}$, where $j'_l < j_{l+1}$ for $l = 1, \ldots, r$ and $j_{r+1} = j$. Further, Lemma 4 shows that these deferred batches may be assumed to appear in σ in the order $\mathscr{B}_1, \ldots, \mathscr{B}_r$. We also assume that $r \leqslant b^2 - b - 1$ in accordance with Lemma 5. If $r \neq 0$, then Lemma 4 shows the batch containing J_{j-1} must be full; therefore, this batch is $\{J_{j-b}, \ldots, J_{j-1}\}$, and $j'_r < j - b$.

We claim that knowledge of the indices of initial jobs $j_1, \ldots, j_r$ allows us to identify, in $O(r)$ time, the indices of final jobs $j'_1, \ldots, j'_r$, and consequently the exact contents of each of the batches $\mathscr{B}_1, \ldots, \mathscr{B}_r$. To justify this claim, we show how to construct $\mathscr{B}_l$, for $l = 1, \ldots, r$. The set $\{J_{j_l}, \ldots, J_{j_{l+1}-1}\}$ comprises batch $\mathscr{B}_l$ and possibly other full batches: except for $\mathscr{B}_l$, there are no non-full batches since the resulting schedule would not be consistent with Lemma 4. Thus, the number of jobs in batch $\mathscr{B}_l$ is $(j_{l+1} - j_l) \bmod b$, from which we deduce that $j'_l = j_l - 1 + (j_{l+1} - j_l) \bmod b$. Since we do not allow empty batches, the choice of initial job indices must satisfy $(j_{l+1} - j_l) \bmod b \neq 0$ for $j = 1, \ldots, r$. We have now established our claim.

We define σ to be in *state* $(j, j_1, \ldots, j_r)$. If $r = 0$, then we write the state as $(j, \circ)$, where $\circ$ is a symbol for empty. A schedule in state $(j, \circ)$ is created by taking one of the following decisions in a previous state:

- *add a batch containing job J_j.* A full or non-full batch $\{J_j, \ldots, J_{k-1}\}$, where $j+1 \leqslant k \leqslant j+b$, is inserted at the beginning of some previous schedule in state $(k, \circ)$.
- *add a full batch that does not contain job J_j.* A full batch $\{J_k, \ldots, J_{k+b-1}\}$, where $j < k \leqslant n-b+1$, is inserted at the beginning of some previous schedule in state $(k+b, \bar{j}_1, \ldots, \bar{j}_{\bar{r}})$, where the corresponding deferred batches $\overline{\mathscr{B}}_1, \ldots, \overline{\mathscr{B}}_{\bar{r}}$ satisfy the relationship $\overline{\mathscr{B}}_1 \cup \cdots \cup \overline{\mathscr{B}}_{\bar{r}} = \{J_j, \ldots, J_{k-1}\}$.

In the latter case, the previous schedule has batches $\overline{\mathscr{B}}_1, \ldots, \overline{\mathscr{B}}_{\bar{r}}$ that are deferred with respect to the batch that contains job J_{k+b-1}. When the full batch $\{J_k, \ldots, J_{k+b-1}\}$ is scheduled, these batches are no longer deferred. Similarly, to create a schedule in state $(j, j_1, \ldots, j_r)$ for $r \neq 0$, we take one of the following decisions:

- *add batch $\mathscr{B}_1$.* Batch $\mathscr{B}_1$ is inserted at the beginning of some previous schedule in state $(j, j_2, \ldots, j_r)$.
- *add a full batch containing job J_j.* A full batch $\{J_j, \ldots, J_{j+b-1}\}$ is inserted at the beginning of some previous schedule in state $(j+b, j_1, \ldots, j_r)$.
- *add a full batch that does not contain job J_j.* A full batch $\{J_k, \ldots, J_{k+b-1}\}$, where $j < k \leqslant n-b+1$, is inserted at the beginning of some previous schedule in state $(k+b, j_1, \ldots, j_r, \bar{j}_1, \ldots, \bar{j}_{\bar{r}})$, where the corresponding deferred batches $\overline{\mathscr{B}}_1, \ldots, \overline{\mathscr{B}}_{\bar{r}}$ satisfy the relationship $\overline{\mathscr{B}}_1 \cup \cdots \cup \overline{\mathscr{B}}_{\bar{r}} = \{J_j, \ldots, J_{k-1}\}$.

Using the above observations, we can use the state variables to compute $j'_1, \ldots, j'_r$ and n', where n' is the number of jobs in σ. We assume that such computations are performed for every state that we consider.

Let $F_j(j_1, \ldots, j_r)$, and $F_j(\circ)$, be the minimum total completion time among all schedules that achieve state $(j, j_1, \ldots, j_r)$ for $r \neq 0$, and $(j, \circ)$, respectively. From our previous observations, we can use the state variables to compute $j'_1, \ldots, j'_r$, the final job indices of deferred batches, and n', where n' is the number of scheduled jobs. We assume that such computations are performed for every state that we consider. In our dynamic programming algorithm, the initialization is

$$F_{n+1}(\circ) = 0$$

For $j = n, n-1, \ldots, 1$, we have recursion equations

$$F_j(\circ) = \min \begin{cases} \min\limits_{j+1 \leqslant k \leqslant j+b} \{F_k(\circ) + n' p_{k-1}\} \\ \min\limits_{j+1 \leqslant k \leqslant n-b+1, (\bar{j}_1, \ldots, \bar{j}_{\bar{r}}) \in H_k} \{F_{k+b}(\bar{j}_1, \ldots, \bar{j}_{\bar{r}}) + n' p_{k+b-1}\} \end{cases}$$

Also, for $j = n+1, n, \ldots, 1$, $r = 1, \ldots, b^2 - b - 1$, and $j_1, \ldots, j_r$ such that $j_1 < \cdots < j_r < j - b$ and $(j_{l+1} - j_l) \bmod b \neq 0$ for $l = 1, \ldots, r$, where $j_{r+1} = j$, the recursion equations are

$$F_j(j_1, \ldots, j_r) = \min \begin{cases} F_j(j_2, \ldots, j_r) + n' p_{j'_1} \\ F_{j+b}(j_1, \ldots, j_r) + n' p_{j+b-1} \\ \min\limits_{j+1 \leqslant k \leqslant n-b+1, (\bar{j}_1, \ldots, \bar{j}_{\bar{r}}) \in H_k} \{F_{k+b}(j_1, \ldots, j_r, \bar{j}_1, \ldots, \bar{j}_{\bar{r}}) + n' p_{k+b-1}\} \end{cases}$$

where H_k is the set of vectors $(\bar{j}_1,\ldots,\bar{j}_{\bar{r}})$ that contain the indices of the initial jobs of deferred batches $\bar{B}_1,\ldots,\bar{B}_{\bar{r}}$, where $1 \leqslant \bar{r} \leqslant b^2 - b - 1 - r$, such that $\bar{B}_1 \cup \cdots \cup \bar{B}_{\bar{r}} = \{J_j,\ldots,J_{k-1}\}$. Note that $\bar{j}_1 = j$. The optimal solution value is equal to $F_1(\circ)$, and the corresponding optimal schedule is found by backtracking.

We now discuss the time and space complexity of our dynamic programming algorithm. There are $O(n^{r+1})$ state variables $(j, j_1,\ldots,j_r)$, and consequently $O(n^{b(b-1)})$ state variables overall. The two terms in the recursion equation for $F_j(\circ)$ are computed in $O(b)$ time and $O(n^{b^2-b-1})$ time, respectively. The first two terms in the equation for $F_j(j_1,\ldots,j_r)$ are computed in constant time, while the third term is computed in $O(n^{b^2-b-1-r})$ time. Thus, the algorithm requires $O(n^{b(b-1)})$ time and $O(n^{b(b-1)})$ space.

We have established that the bounded problem of minimizing the total completion time is polynomially solvable for fixed b. However, when b is arbitrary, the complexity remains unresolved.

4.1.2. The case of m different processing times

In this subsection, we present an $O(b^2m^2 2^m)$ time dynamic programming algorithm for minimizing total completion time for the case of m different job processing times. Before proceeding, we introduce some notation that is specifically needed for the development of this algorithm. Let the distinct processing times be $\bar{p}_1 < \cdots < \bar{p}_m$, and let $\mathscr{S}_j$ be the set of all jobs with processing time equal to $\bar{p}_j$, for $j = 1,\ldots,m$. Moreover, let $b_j = \lfloor |\mathscr{S}_j|/b \rfloor$, for $j = 1,\ldots,m$, so that b_j represents the maximum number of full batches containing jobs in $\mathscr{S}_j$ only. We call a job a *j-job* if its processing time is $\bar{p}_j$, and we call a batch a *j-batch* if its longest job is a j-job. We call a j-batch *pure* if the batch is full and contains j-jobs only; otherwise, it is *non-pure*.

In accordance with Lemmas 3 and 4, we restrict our search to schedules in which batches are sequenced in order of non-decreasing ratios $p(\mathscr{B}_l)/|\mathscr{B}_l|$, and no batch is deferred with respect to a non-full batch. Instead of using Lemma 2, however, we characterize a set of optimal solutions by properties stipulated in the following lemma.

Lemma 6. There exists an optimal schedule with the following properties:

(i) (Chandru *et al.* [8]) *there are b_j pure j-batches, for $j = 1,\ldots,m$;*
(ii) (Hochbaum and Landy [9]) *there is at most one non-pure j-batch, for $j = 1,\ldots,m$.*

Property (i) states that there are b_j pure j-batches, for $j = 1,\ldots,m$, which from Lemma 3 must be sequenced in order of non-decreasing $\bar{p}_j$. For the remaining jobs, we enumerate all possible configurations of full non-pure batches. Since property (ii) shows that we may restrict our attention to schedules that have either zero or one full non-pure j-batch for each j ($j = 1,\ldots,m$), we need to consider 2^m possible combinations. We represent a given configuration of full non-pure batches by the set of indices $\mathscr{X} \subset \{1,\ldots,m\}$, where $j \in \mathscr{X}$ if and only if there is a full non-pure j-batch. We introduce indicator variables $a_{j,\mathscr{X}}$, where

$$a_{j,\mathscr{X}} = \begin{cases} 1 & \text{if } j \in \mathscr{X} \\ 0 & \text{otherwise} \end{cases}$$

Given a set $\mathscr{X}$, we propose a batch filling procedure that finds all other jobs in the corresponding full non-pure batches. Consider any index j, where $j \in \mathscr{X}$. Since a j-batch is required to have $\bar{p}_j$ as its processing time, it must contain a j-job and be filled with jobs with processing times at most $\bar{p}_j$. Apart from this upper bound on the processing time, we are free to assign any other $b - 1$

jobs, although it is best to fill the batch with jobs having processing times as large as possible. This observation leads to the batch filling procedure described below.

Batch filling procedure

Input. Any set $\mathscr{X}$ of batch indices.

Step 0. Initially, set $n_{j,\mathscr{X}} = |\mathscr{S}_j| - b_j b$ if $j \notin \mathscr{X}$, and $n_{j,\mathscr{X}} = |\mathscr{S}_j| - b_j b - 1$ if $j \in \mathscr{X}$, for $j = 1,\ldots,m$. Note that $n_{j,\mathscr{X}}$ represents the number of j-jobs that remain to be assigned to the non-pure j-batches. Set $h = 0$.

Step 1. Determine the smallest index j such that $j \in \mathscr{X}$ and $j > h$. If no such index exists, then terminate the procedure with an optimal set of full non-pure batches.

Step 2. If $\sum_{i=1}^{j} n_{i,\mathscr{X}} < b-1$, then terminate the procedure: it is not possible to fill all non-pure batches specified by $\mathscr{X}$.

Let i be the largest index $i < j$ for which

$$n_{i,\mathscr{X}} + n_{i+1,\mathscr{X}} + \cdots + n_{j-1,\mathscr{X}} + n_{j,\mathscr{X}} \geqslant b - 1 \tag{10}$$

Fill the non-pure j-batch with all unassigned jobs from the sets $\mathscr{S}_{i+1},\ldots,\mathscr{S}_j$ and with $b - 1 - \sum_{k=i+1}^{j} n_{k,\mathscr{X}}$ unassigned i-jobs. Accordingly, we update $n_{i,\mathscr{X}} = n_{i,\mathscr{X}} - b + 1 + \sum_{k=i+1}^{j} n_{k,\mathscr{X}}$ and $n_{k,\mathscr{X}} = 0$ for $k = i+1,\ldots,j$.

Step 3. Set $h = j$, and go to Step 1.

This procedure can be implemented to run in $O(m)$ time for any given set $\mathscr{X}$.

The remaining issue is how to form non-full batches with the remaining jobs, which have not been assigned by the batch filling procedure, and how to interleave these non-full batches with the full batches to minimize total completion time. For ease of exposition, assume that the remaining jobs are $J_1,\ldots,J_{n'}$, where these jobs have been indexed according to the SPT rule. Let p_j be the processing time of J_j, for each $j = 1,\ldots,n'$. Note that $n' \leqslant m(b-1)$. The following result establishes that there exists an optimal schedule in which, after removal of all full batches, jobs $J_1,\ldots,J_{n'}$ form an SPT-batch schedule.

Lemma 7. There exists an optimal schedule in which the sequence of batches formed by jobs $J_1,\ldots,J_{n'}$ is $(\mathscr{B}_1,\ldots,\mathscr{B}_r)$, where under the SPT indexing $\mathscr{B}_l = \{J_{j_l},J_{j_l+1},\ldots,J_{j_{l+1}-1}\}$ and $j_{l+1} < j_l + b$, for $j = 1,\ldots,r$, and $1 = j_1 < j_2 < \cdots < j_r < j_{r+1} = n' + 1$.

Proof. The argument used in the proof of Lemma 2 shows that there exists an optimal schedule in which the non-full batches contain jobs with consecutive indices. Lemma 4 establishes that these batches are sequenced in order of increasing processing times. ∎

We can restrict our search for an optimal schedule to one which has the following properties:

(a) the full batches appear in order of non-decreasing processing times, which follows from Lemma 3;
(b) the remaining jobs form non-full batches, where each such batch contains jobs with consecutive indices, and the batches are sequenced in SPT order, which follows from Lemma 7;
(c) a complete schedule is found by interleaving the full and non-full batches.

If the non-full batches are known, the interleaving can be performed using Lemma 3. However, we propose a dynamic programming algorithm that forms the non-full batches and interleaves the full and non-full batches.

For any given $\mathscr{X}$, we apply a backward dynamic programming algorithm in which complete batches, either full or non-full, are added to the beginning of some previous schedule. This algorithm implicitly enumerates all schedules that have properties (a) and (b). Since we are minimizing total completion time, it will automatically give an optimal schedule that possesses property (c) as well.

Let σ be a schedule that satisfies properties (a) and (b), that contains all of the full k-batches, for $k = i,\ldots,m$, and the jobs $J_j,\ldots,J_{n'}$ in non-full batches. Processing of the first batch starts at time zero. We define such a schedule to be in state (i,j). Let $F_i(j)$ be the minimum total completion time for schedules in state (i,j). A schedule in state (i,j) is created by taking one of the following decisions in a previous state:

(1) *add all full i-batches.* The number of full i-batches is equal to $b_i + a_{i,\mathscr{X}}$ (which may be zero). Inserting these batches at the beginning of some previous schedule in state $(i+1,j)$ delays the jobs that are already scheduled by $(b_i + a_{i,\mathscr{X}})\bar{p}_i$ units, and therefore increases the total completion time by $(b_i + a_{i,\mathscr{X}})(b_i + a_{i,\mathscr{X}} + 1)b\bar{p}_i/2 + (n' - j + 1 + b\sum_{k=i+1}^{m}(b_k + a_{k,\mathscr{X}}))(b_i + a_{i,\mathscr{X}})\bar{p}_i$.

(2) *add a batch containing J_j.* A non-full batch $\{J_j,\ldots,J_{k-1}\}$, where $j < k \leqslant \min\{n'+1, j+b-1\}$, is inserted at the beginning of some previous schedule in state (i,k). Since this batch has processing time p_{k-1}, the previously scheduled jobs are completed p_{k-1} units later when this batch is added, and total completion time increases by $(n'-j+1+b\sum_{k=i}^{m}(b_k+a_{k,\mathscr{X}}))p_{k-1}$.

We are now ready to give the dynamic programming algorithm for computing an optimal schedule for any given $\mathscr{X}$. The initialization is

$$F_{m+1}(n'+1) = 0$$

and the recursion for $i = m+1, m, \ldots, 1$ and $j = n'+1, n', \ldots, 1$ (where either $i < m+1$ or $j < n'+1$) is

$$F_i(j) = \min \begin{cases} F_{i+1}(j) + (b_i + a_{i,\mathscr{X}})(b_i + a_{i,\mathscr{X}} + 1)b\bar{p}_i/2 \\ \qquad + \left(n' - j + 1 + b\sum_{k=i+1}^{m}(b_k + a_{k,\mathscr{X}})\right)(b_i + a_{i,\mathscr{X}})\bar{p}_i \\ \min_{j<k\leqslant \min\{n'+1, j+b-1\}} \{F_i(k) + \left(n' - j + 1 + b\sum_{k=i}^{m}(b_k + a_{k,\mathscr{X}})\right) p_{k-1}\} \end{cases}$$

The optimal solution value for a given set $\mathscr{X}$ is $F_1(1)$ and the corresponding optimal schedule is found by backtracking.

To implement the algorithm efficiently, the partial sums $\sum_{k=i}^{m}(b_k + a_{k,\mathscr{X}})$ are evaluated and stored for $i = 1,\ldots,m$ in a preprocessing step in $O(m)$ time. Then, each value $F_i(j)$ is determined in $O(b)$ time. Since $n' \leqslant (m-1)b$, the recursion requires $O(b^2m^2)$ time and $O(bm^2)$ space altogether for a given $\mathscr{X}$. To solve the problem, we need to run the recursion for all feasible sets $\mathscr{X}$, of which there are at most 2^m. Hence, the overall time requirement for the algorithm is $O(b^2m^2 2^m)$, and the space requirement is $O(bm^2)$.

4.2. Scheduling with due dates is NP-hard

In this section, we prove that bounded problems with due dates or deadlines are NP-hard in the strong sense, even if $b = 2$.

Theorem 3. Finding a feasible solution to a bounded problem with deadlines is unary NP-complete, even if $b = 2$.

Proof. The proof is based upon a reduction from the unary NP-complete problem 3-PARTITION.

3-PARTITION

Given a set $\{a_1, \ldots, a_{3m}\}$ of $3m$ positive integers with $\sum_{j=1}^{3m} a_j = mA$ and $A/4 < a_j < A/2$ for $j = 1, \ldots, 3m$, is it possible to partition the index set $\{1, \ldots, 3m\}$ into m mutually disjoint subsets $\mathscr{X}_1, \ldots, \mathscr{X}_m$ such that $\sum_{i \in \mathscr{X}_j} a_i = A$ for $j = 1, \ldots, m$?

Given an instance of 3-PARTITION, we construct the following instance of our bounded deadlines problem with $6m^2$ jobs and $b = 2$. For $i = 1, \ldots, 3m$ and $j = 0, \ldots, m$, there is a job $J_{i,j}$ with processing time $p_{i,j}$ and deadline $\bar{d}_j$ for $j \neq 0$ and deadline $\bar{d}_m$ for $j = 0$, where

$$p_{i,j} = iW + (m - j)a_i$$

$$\bar{d}_j = \sum_{h=1}^{j} \left(\sum_{i=1}^{3m} iW + (m - h)mA + hA \right)$$

and $W = m^3A$. Also, for $i = 1, \ldots, 3m$ and $j = 2, \ldots, m$, there is a job $J_{3m+i,j}$ with processing time $p_{3m+i,j} = p_{i,j}$ and deadline $\bar{d}_j$. Any job $J_{i,j}$ for $j = 0, \ldots, m$, and $J_{3m+i,j}$ for $j = 2, \ldots, m$ is of *type i*. Further, the jobs $J_{i,j}$, and the jobs $J_{3m+i,j}$ (for $j \neq 0, 1$), for $i = 1, \ldots, 3m$, form *group j*. Note that groups 0 and 1 each contain $3m$ jobs, while all other groups contain $6m$ jobs.

We first show that the reduction for constructing the instance of the bounded deadlines problem is polynomial. Under a unary encoding, the size of the input for 3-PARTITION is mA. For the bounded deadlines problem, upper bounds on the total size of each processing time and each deadline under a unary encoding are $3mW + mA$ and $m(9m^2W + m^2A + mA)$, respectively. Since there are $6m^2$ jobs, and $W = m^3A$, the total size of the instance is bounded above by $6m^2(3m^4A + mA + 9m^6A + m^3A + m^2A)$, which is polynomial in m and A. Thus, our reduction is polynomial.

In the remainder of the proof, we show that 3-PARTITION has a solution if and only if there is a feasible solution to this scheduling instance with deadlines.

First, suppose that $\mathscr{X}_1, \ldots, \mathscr{X}_m$ defines a solution to 3-PARTITION. Consider a schedule which comprises m blocks of batches, where each block contains $3m$ batches. In block j, for $j = 1, \ldots, m$, there are three batches $\{J_{i,j}, J_{i,0}\}$ for $i \in \mathscr{X}_j$ and $3m - 3$ batches $\{J_{i,j}, J_{3m+i,j}\}$ (for $j \neq 1$) or $\{J_{i,j}, J_{3m+i,j+1}\}$, depending on whether job $J_{3m+i,j}$ (for $j \neq 1$) is scheduled in a previous block, for $i \notin \mathscr{X}_j$. Note that all jobs in group 1 are scheduled in block 1, and all jobs in group j, for $j = 2, \ldots, m$, are scheduled either in block $j - 1$ or block j. The completion of processing of block j occurs at time

$$\sum_{h=1}^{j} \left(\sum_{i=1}^{3m} iW + \sum_{i \in \mathscr{X}_h} ma_i + \sum_{i \notin \mathscr{X}_h} (m - h)a_i \right)$$

which can be expressed as

$$\sum_{h=1}^{j}\left(\sum_{i=1}^{3m} iW + (m-h)mA + hA\right) = \bar{d}_j$$

Therefore, each job is completed by its deadline, and the schedule is feasible.

Conversely, suppose that there exists a feasible schedule. We show first that each batch contains exactly two jobs which are of the same type. If this is not the case, then the completion time T of the last batch in the schedule satisfies

$$T > \sum_{h=1}^{m}\sum_{i=1}^{3m} iW + W$$

Using the inequality $(m-h)mA + hA \leqslant m^2A$ in the expression for $\bar{d}_m$ yields

$$\bar{d}_m \leqslant \sum_{h=1}^{m}\sum_{i=1}^{3m} iW + m^3A$$

which shows that $T > \bar{d}_m$, and the schedule is not feasible.

Jobs $J_{i,1}$, for $i = 1,\ldots,3m$, are each contained in different batches which are completed by time $\bar{d}_1$. Since $\sum_{i=1}^{3m} p_{i,1} > \sum_{i=1}^{3m} iW = \bar{d}_1 - (m-1)mA - A \geqslant d_1 - W$, these $3m$ batches must be sequenced in the first $3m$ positions of the schedule. It is not possible for both $J_{i,2}$ and $J_{3m+i,2}$ to be scheduled in the first block of $3m$ batches. Since $J_{i,2}$ and $J_{3m+i,2}$ are identical, we may assume that jobs $J_{2,i}$, for $i = 1,\ldots,3m$, are each contained in different batches which are not sequenced in the first $3m$ positions. To achieve the deadline $\bar{d}_2$, these batches must form a second block which are sequenced in positions $3m+1,\ldots,6m$. Repetition of this argument shows that there is a block of $3m$ batches which contain jobs $J_{i,j}$ for $i = 1,\ldots,3m$ that are sequenced in positions $3(j-1)m+1,\ldots,3jm$, respectively, for $j = 1,\ldots,m$. Moreover, the batch containing job $J_{i,j}$ also contains either $J_{i,0}$, $J_{3m+i,j}$ or $J_{3m+i,j+1}$ (for $j \neq m$).

The batch containing job $J_{i,j}$ has processing time $p_{i,j}$ unless it contains job $J_{i,0}$ in which case it has processing time $p_{i,0}$. We identify the batches to which jobs $J_{1,0},\ldots,J_{3m,0}$ are assigned in order to define a partition $\mathscr{X}_1,\ldots,\mathscr{X}_m$ of the index set $\{1,\ldots,3m\}$. Specifically, $i \in \mathscr{X}_j$ if $\{J_{i,j},J_{i,0}\}$ is a batch. Since all batches containing jobs $J_{i,j}$ for $i = 1,\ldots,3m$ must be completed by time $\bar{d}_j$, we obtain

$$\sum_{h=1}^{j}\left(\sum_{i\notin\mathscr{X}_h} p_{i,h} + \sum_{i\in\mathscr{X}_h} p_{i,0}\right) \leqslant \bar{d}_j, \quad j = 1,\ldots,m$$

which can be expressed as

$$\sum_{h=1}^{j} h\left(A - \sum_{i\in\mathscr{X}_h} a_i\right) \geqslant 0, \quad j = 1,\ldots,m \tag{11}$$

Suppose that at least one of the inequalities in (11) is strict. Then by forming a linear combination of the m inequalities with positive coefficients $1/j - 1/(j+1)$ for $j = 1,\ldots,m-1$, and $1/m$ for $j = m$, we obtain a strictly positive value. This relationship is expressed as

$$\sum_{j=1}^{m-1}\left(\frac{1}{j} - \frac{1}{j+1}\right)\sum_{h=1}^{j} h\left(A - \sum_{i\in\mathscr{X}_h} a_i\right) + \frac{1}{m}\sum_{h=1}^{m} h\left(A - \sum_{i\in\mathscr{X}_h} a_i\right) > 0$$

which can be rewritten as

$$\sum_{h=1}^{m} h\left(A - \sum_{i\in\mathscr{X}_h} a_i\right)\left(\sum_{j=h}^{m-1}\left(\frac{1}{j} - \frac{1}{j+1}\right) + \frac{1}{m}\right) = mA - \sum_{h=1}^{m}\sum_{i\in\mathscr{X}_h} a_i > 0$$

However, this inequality contradicts $\sum_{i=1}^{3m} a_i = mA$, which is obtained from the definition of 3-PARTITION. Therefore, the left-hand side of each inequality in (11) is equal to zero. This implies that $\sum_{i\in\mathscr{X}_j} a_i = A$ for $j = 1,\ldots,m$, which shows that $\mathscr{X}_1,\ldots,\mathscr{X}_m$ define a solution to 3-PARTITION ■

Corollary 1. The bounded problems of minimizing the maximum lateness $L_{\max}$, *the number of tardy jobs* $\sum_{j=1}^{n} U_j$, *and the total tardiness* $\sum_{j=1}^{n} T_j$ *are unary NP-hard, even if* $b = 2$.

4.3. Restricted number of batches

In this section, we consider the bounded problem of minimizing any regular objective function when the schedule is constrained to contain at most r batches. We show that this problem can be solved in $O(n^{r+3})$ time, which is a polynomial when r is fixed.

Note that, if the longest job of each batch is specified along with the order in which the r batches are sequenced, then it is straightforward to compute the processing times $p(\mathscr{B}_1),\ldots,p(\mathscr{B}_r)$ and completion times $C(\mathscr{B}_1),\ldots,C(\mathscr{B}_r)$ of the batches. The problem then reduces to one of assigning each of the remaining jobs to the r batches so that no batch contains more than b jobs and no job is assigned to a batch whose designated longest job would be smaller. If job J_j has a deadline $\bar{d}_j$, then we must also ensure that $C_j \leqslant \bar{d}_j$.

From the above observations, the cost c_{ij} of assigning any of the $n - r$ remaining jobs J_j $(j = 1,\ldots,n)$ to batch $\mathscr{B}_i$ $(i = 1,\ldots,r)$ is

$$c_{ij} = \begin{cases} \infty & \text{if } p(\mathscr{B}_i) < p_j \text{ or } C(\mathscr{B}_i) > \bar{d}_j \\ f_j(C(\mathscr{B}_i)) & \text{otherwise} \end{cases}$$

Thus, the problem of minimizing a minsum cost function for given batch processing times and a given processing order of the r batches reduces to a bipartite weighted matching problem, which can be solved in $O(n^3)$ time [15].

To select the designated longest jobs, each of the $\binom{n}{r}$ possible choices must be considered. For each selection, there are $r!$ batch sequences. For fixed r, there are $O(n^r)$ selections of longest jobs and batch processing orders, each of which requires a bipartite weighted matching problem to be solved. Thus, the problem is solvable in $O(n^{r+3})$ time, which is polynomial.

To minimize a regular minmax objective function, we adopt a similar approach, except that it is necessary to solve a minmax bipartite weighted matching problem. Since this matching problem is solvable in $O(n^3)$ time [15], the overall time requirement for a minmax objective function is also $O(n^{r+3})$.

5. CONCLUDING REMARKS

This paper is a first step towards providing a complexity mapping of single-machine batching problems, in which the processing time of a batch is dictated by its longest job. We refer to Table 1 in the introduction for a summary of our results. The mapping is not complete, since the following complexity issues remain open: binary NP-hardness for the unbounded problem of

minimizing the total tardiness $\sum_{j=1}^{n} T_j$ (which is pseudopolynomially solvable), and binary and unary NP-hardness for the bounded problems of minimizing the total completion time $\sum_{j=1}^{n} C_j$ for arbitrary b and of minimizing the total weighted completion time $\sum_{j=1}^{n} w_j C_j$ for fixed and arbitrary b.

Our analysis does not consider problems with unequal job release dates. For the unbounded model, a different approach to that in Section 3 is required, since we may no longer restrict our search to SPT-batch schedules. These problems are therefore likely to be much more difficult than their counterparts with equal job release dates. There is one exception: since the problem of minimizing the makespan subject to unequal job release dates is the mirror image of minimizing the maximum lateness with equal release dates, we can solve the makespan problem in $O(n^2)$ time using the algorithm for minimizing the maximum lateness that is presented in Section 3.5.

For the bounded model with unequal release dates, all criteria that we consider give rise to unary NP-hard problems. Table 1 shows that most such problems are already unary NP-hard with equal release dates, while minimizing the total completion time subject to release dates is unary NP-hard even if $b = 1$ [16]. Minimizing the makespan subject to release dates is unary NP-hard as well, since we have shown in Section 4.2 that the equivalent mirror image problem of minimizing the maximum lateness is unary NP-hard for $b = 2$.

Another extension to the model involves scheduling jobs with equal release dates on m identical parallel machines. For the unbounded model, it is not difficult to prove that there exists an optimal solution which is an SPT-batch schedule for an arbitrary regular objective function. We claim therefore that any such parallel-machine batching problems for which the SPT-batch property still holds can be solved by dynamic programming in pseudopolynomial time for a *fixed* number of machines.

ACKNOWLEDGEMENTS

The research reported in this paper was supported by INTAS (Project INTAS-93-257 and INTAS-93-257-Ext). Thomas Tautenhahn has been supported by Deutscher Akademischer Austauschdienst.

REFERENCES

1. C.-Y. Lee, R. Uzsoy and L. A. Martin-Vega, 'Efficient algorithms for scheduling semiconductor burn-in operations', *Oper. Res.*, **40,** 764–775 (1992).
2. S. Webster and K. R. Baker, 'Scheduling groups of jobs on a single machine', *Oper. Res.*, **43,** 692–703 (1995).
3. S. Albers and P. Brucker, 'The complexity of one-machine batching problems', *Discrete Appl. Math.*, **47,** 87–107 (1993).
4. J. H. Ahmadi, R. H. Ahmadi, S. Dasu and C. S. Tang, 'Batching and scheduling jobs on batch and discrete processors', *Oper. Res.*, **39,** 750–763 (1992).
5. R. Uzsoy, 'Scheduling batch processing machines with incompatible job families', *Int. J. Prod. Res.*, **33,** 2685–2708 (1995).
6. T. C. E. Cheng and M. Y. Kovalyov, Addendum to C.-Y. Lee, R. Uzsoy and L. A. Martin-Vega (1992), Working Paper, Institute of Engineering Cybernetics, Minsk, Belarus, 1995.
7. V. Chandru, C.-Y. Lee and R. Uzsoy, 'Minimizing total completion time on batch processing machines', *Int. J. Prod. Res.*, **31,** 2097–2122 (1993).
8. V. Chandru, C.-Y. Lee and R. Uzsoy, 'Minimizing total completion time on a batch processing machine', *Oper. Res. Lett.*, **13,** 61–65 (1993).
9. D. S. Hochbaum and D. Landy, 'Scheduling semiconductor burn-in operations to minimize total flowtime', *Oper. Res.*, **45,** 874–885 (1997).
10. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, 'Sequencing and scheduling: algorithms and complexity', in S. C. Graves, P. H. Zipkin and A. H. G. Rinnooy Kan (eds.), *Logistics of Production and Inventory; Handbooks in Operations Research and Management Science, Vol. 4*, North-Holland, Amsterdam, 1993, pp. 445–522.

11. G. J. Woeginger, 'When does a dynamic programming formulation guarantee the existence of an FPTAS?', *Technical Report Woe-27*, TU-Graz, Austria, 1998.
12. S. Van Hoesel, A. Wagelmans and B. Moerman, 'Using geometric techniques to improve dynamic programming algorithms for the economic lot-sizing problem and extensions', *European J. Oper. Res.*, **75,** 312–331 (1994).
13. N. Blum, R. W. Floyd, V. Pratt, R. L. Rivest and R. E. Tarjan, 'Time bounds for selection', *J. Comput. Systems Sci.*, **7,** 448–461 (1973).
14. A. Schonhäge, M. Patterson and N. Pippenger, 'Finding the median', *J. Comput. Systems Sci.*, **13,** 184–199 (1976).
15. E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.
16. J. K. Lenstra, A. H. G. Rinnooy Kan and P. Brucker, 'Complexity of machine scheduling problems', *Ann. Discrete Math.*, **1,** 343–362 (1977).

